OGI Spoken Term Detection System

Zak Shafran Brian Roark Seeger Fisher
OGI School of Science & Engineering, Beaverton, OR 97006
{zak,roark,fishers}@cslu.ogi.edu

Acknowledgements: Dimitra Verygri & Andreas Stolcke, SRI

Dec 14, 2006

Key Features

- FSM-based implementation: flexible and quick prototyping
- Word-level tokens
- Two-stage search; allowing more complex models
- OOV & text-normalization
- Rescoring paradigm
- Query expansion

Transducer Index

1. Input: ASR word lattices from SRI
2. First, we compute forward-backward algorithm to convert the likelihood scores into posteriors
3. Since acoustic scores have large dynamic range, likelihoods are squashed by dividing it with language model factor before computing posteriors, a practice fairly common in ASR
4. Resulting lattices are pruned to eliminate paths with low scores
5. Finally, word posterior lattices are converted into transducer index

Transducer Index

- Input: All *n*-gram sequences present in ASR lattices
- Output: Utterance IDs associated with each *n*-gram sequence
- Cost: Posterior probability of the sequence in the utterance

Steps to Create Transducer Index

1. ASR lattice of each utterance is turned into an *n*-gram FSM (Allauzen et al 2004)
2. 5-word query length restriction is applied by composing the *n*-gram FSM with a constraint automaton
3. Output labels of final transitions are tagged with utterance ID
4. Optimization by applying weighted ϵ-removal, weighted determinization and minimization over the log semiring, after encoding the transducer as an equivalent acceptor
5. Transducer index, $\mathcal{I}$, is created by the union of all utterance-level transducers

Baseline Search

1. Create a query FSM, q
2. Compose query FSM with FST index, $q \circ \mathcal{I}$
3. Pick top 1000 hits, namely, fsmbestpath -n 1000
4. For each utterance, return the times of best occurrence

Other Variants

Transducer index can be converted into phone-based index and search can easily be performe at phone level

1. Expand the query into its pronunciation, $q\mathcal{L}$
2. Compose query FSM with phone-level FST index, $q\mathcal{L} \circ \mathcal{L} \circ \mathcal{I}$

   ...

Baseline Results

	ATWV			
Test set	Total	BN	CTS	MTG
2006 Dev	**0.769**	0.833	0.673	0.256
2006 Dry	**0.678**	0.777	0.598	0.196

Performance: BN $\gg$ CTS $\gg$ MTG Results on Dev $\gg$ Dry

Types of Errors?

1. How much can BN / CTS / MTG benefit from fixing false alarms?
2. How much can BN / CTS / MTG benefit from fixing misses?

Oracle Experiment

1. Perform an exhaustive search and score the result
2. Using the alignment (csv) and judgements (corr/fa) remove false alarms to create an oracle result
3. Score the oracle result

Oracle Score

	ATWV			
Test	Total	BN	CTS	MTG
Dev	**0.827** (0.769)	0.874 (0.833)	0.813 (0.673)	0.617 (0.256)
Dry	**0.757** (0.678)	0.834 (0.777)	0.759 (0.598)	0.572 (0.196)

Observations

- Eliminating false alarms improves CTS & MTG significantly!
  - Adopt a rescoring paradigm to reduce false alarms
- Misses accounts for the gap from unity. MTG particularly suffers from it!
  - OOV & Text-normalization to recover misses

Rescoring Paradigm

- Key idea: reduce false alarms
- SVM-based rescoring (*libsvm*)
- Features:
  1. Number of words
  2. Number of phones
  3. Type of utterance (BN/CTS/MTG)
  4. *n*-gram posterior
  5. Density of lattice: #states, #arcs
  6. Frequent vs infrequent word
- Train of dev, test on dry and vice versa

Rescoring Results

	ATWV			
Test	Total	BN	CTS	MTG
Dev	**0.680**	0.779	0.598	0.189
Dry	**0.769**	0.834	0.673	0.262

Conclusions

1. Training and testing (cheating experiments) on Dev & Dry show substantial improvements
2. However, only marginal generalization across sets, possibly, due to bad features or small size of the held-out set
3. Additional held-out data set was created using the NIST scripts, but its composition differed from the NIST set and did not help
4. So, in the end, Dev and Dry data was pooled to train the rescoring algorithm and was used in the evaluation

OOV & Text-normalization

- Non-standard spelling in query can be problematic
  - e.g., ‘Hanson’ versus ‘Hanssen’ (NIST 2006 STD Dev)
- ASR transcripts may contain orthographic variants of differing frequency
  - ‘Mr.’ versus ‘Mr’ (NIST 2006 STD Dev)
- Queries with true OOVs maybe recovered by mapping them to similar sounding words from ASR vocabulary
  - ‘billionaire’ versus ’billion air’
- A single approach was developed to handle the above problems

OOV & Text-normalization (contd.)

- Build a word transducer $\mathcal{L}$
- Estimate the expected unigram word frequency $\mathcal{G}$ in the indexed ASR lattices
- Compose the two, $\mathcal{L} \circ \mathcal{G}$ to provide weighted transducer $\mathcal{LG}$
- At test time, expand each query word to its pronunciation, $q\mathcal{L}$, which is then composed with $\mathcal{LG}$, $q\mathcal{L} \circ \mathcal{LG}$, and projected onto outputs
- If non-empty, word is replaced with highest scoring output
  - Unless pronunciation has less than 4 phones
- To handle pronunciations of unseen words, Festival was used

Results OOV & Text-normalization

	ATWV			
Test	Total	BN	CTS	MTG
Dev	**0.777** (0.769)	0.843 (0.833)	0.674 (0.673)	0.264 (0.256)
Dry	**0.692** (0.678)	0.795 (0.777)	0.594 (0.598)	0.143 (0.196)

Small consistent improvements

In the works, but didn't make it

1. System combination with SRI
2. Phone-based and mixed-token system
3. Query expansion
4. Learned OOV & text-normalization

Query Expansion

- boost scores for *Vatican* when it co-occurs with *Rome*
- for each query, look up Gigaword corpus and come up with a candidate set of "good" co-occurring words
- test the present of these telltale words in the retrieved utterance
- new scores computed using the same rescoring paradigm mentioned earlier